

4

AD-A203 608

DTIC FILE COPY

Equipment Simulation for Language Understanding

Tomasz Ksiezyk and Ralph Grishman
PROTEUS Project Memorandum #11-A
June 1988

DTIC
SELECTE
JAN 27 1989
S D⁶⁶ D

To appear in The International Journal of Expert Systems

DISSEMINATION STATEMENT A
Approved for public release;
Distribution Unlimited

This report is based upon work supported by the
Defense Advanced Research Projects Agency under Contract
N00014-85-K-0163 from the Office of Naval Research, and
by the National Science Foundation under grant DCR-85-01843.

89 1 10 048

Equipment Simulation for Language Understanding

Tomasz Ksiezyk
Ralph Grishman
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
(212) 998-3491, ksiezyk@acf5.nyu.edu

20 September 1987
Revised 20 June 1988

Abstract

We consider in this paper the task of analyzing reports regarding the failure, diagnosis and repair of equipment. We show that a general knowledge of equipment is not sufficient for a full understanding of such reports. As an alternative, we describe a system — PROTEUS — which relies on a detailed simulation model to support language understanding. We describe the structure of the model and emphasize features specifically required for language understanding. We show how this model can be used in analyzing and determining the referents for complex noun phrases. We point out the importance of identifying the implicit temporal and causal relations in the text and show how the simulation capabilities of the model support this task. Finally, we indicate how PROTEUS may be extended to facilitate the entry of new equipment models and to assist in equipment diagnosis.

1 Introduction

The work presented here is part of PROTEUS (PROtotype Text Understanding System), currently under development at the Courant Institute of Mathematical Sciences, New York University.¹ The objective of our research is to understand short natural language texts about equipment. Our texts at present are CASualty REPorts (CASREPs) which describe failures of equipment installed on Navy ships. Our initial domain is the starting air system for propulsion gas turbines. A typical CASREP consists of several sentences, for example:²

Unable to maintain lube oil pressure to SAC (Starting Air Compressor). Disengaged immediately after alarm. Metal particles in oil sample and strainer.

It is widely accepted among researchers that in order to create natural language understanding systems robust enough for practical application, one must provide them with a lot of common-sense and domain-specific knowledge. However, so far, there is no consensus as to the best way of choosing, organizing and using such knowledge.

The novelty of the approach presented here is that, besides general knowledge about equipment, we also use a quite extensive simulation model for the specific piece of equipment which the texts deal with. We see the following merits of having a simulation model:

¹ This work is being done in collaboration with UNISYS as part of the DARPA Strategic Computing Program.

² Unless noted otherwise, all examples given in this paper are from actual CASREPs.



J	
□	
□	
per ltr	
Dist	100
A-1	

- The model provides us with a reliable background against which we can check the correctness of the understanding process on several levels: finding referents of noun phrases, assigning semantic cases to verbs, establishing causal relationships between individual sentences of the text.
- The requirements of simulation help us to decide what kind of knowledge about the equipment should be included in the model, how it could best be organized and which inferences it should be possible to make. It appears that the information needed for simulation largely coincides with that necessary for language understanding.
- The ability to simulate the behavior of a piece of equipment provides an effective means of verifying the system's understanding of a message: it is relatively straightforward to build a dynamic graphical interface which gives the user insight into the way his input has been understood by the system.

In addition to its role in language understanding, we believe that the simulation model provides many — perhaps most — of the capabilities which would be needed for a model-based diagnostic or tutorial system.

We can compare our approach to model building with that often followed in application systems with natural language front ends, such as data base, diagnostic, and tutorial systems. In most cases such front ends were conceived as add-ons to existing systems, often only after such systems were in use for some time. This leads to a situation where much of the knowledge necessary to solve some understanding problems already existed in the main system, but couldn't be used by the natural language module because its structure had been designed without regard to the needs of language analysis. As a result a separate knowledge base had to be built for the language module, duplicating to a large extent the knowledge in the base system, and the two knowledge bases had to be interfaced, complicating the overall design.

We have in contrast focussed from the outset on the needs of natural language understanding. We have been led to develop a detailed simulation model, as would be required for a model-based diagnostic or tutorial system. We have developed a rich graphical interface, which would be of benefit for these other applications too. But we have also incorporated features which were specifically dictated by the needs of language understanding, such as a mixed static/dynamic model and the ability to refer to dynamically-created aggregates. We believe that this conceptually richer model, suitably extended, would provide a good basis for a model-based expert system with an *integrated* natural language interface.

In a language system incorporating such a simulation model, the analysis of a piece of text involves the following steps:

1. locating in the model the objects and other concepts mentioned in text (they are usually referred to by noun phrases) and creating internal representations for those objects and other concepts which are referred to in the text but are not part of the model;
2. recognizing elementary facts reported in the text and building their representations; these facts are either about the above objects and concepts or about other facts;
3. finding relationships between elementary facts which make them into a coherent whole; such relationships may be given explicitly in the text or may have to be inferred; the inference mechanism is based largely on queries to the model and on requests for the simulation of certain facts.

In the remainder of this paper we shall address the following issues: Why is a detailed equipment model needed? How should such a model be structured and, in particular, what balance should we strike between a static model and one created dynamically as the text requires? How should a noun phrase analyzer be organized to utilize such a model, and how is it affected by this static/dynamic balance? How does the discourse analysis module work and what role does the simulation play in it? What are the ways of supporting users in designing new equipment models?

2 The Need for a Model

In many natural language understanding systems the knowledge about the domain of discourse is organized in the form of prototypes for objects and actions, and for the relations between them which are relevant for the domain. The prototypes are repositories for general knowledge about the instances they may subsume. As each sentence is processed, it is split into units which are identified as possible instances of prototypes in the knowledge base. Through these prototypes access to general information about the concepts invoked by the sentence is achieved. This information is often necessary for the adequate interpretation (i.e. understanding) of the sentence. To account for the fact that the understanding of an utterance depends sometimes on context, it is necessary to maintain information about the discourse context. One way of organizing this information is by creating and storing instances of prototypes for entities from the text as they are analyzed. The combined information coming from the context and from the processed sentence is used to solve problems like anaphora resolution, connectivity, etc.

Assuming this approach, let's consider the following sentence (let it be the first sentence in the analyzed text):³

Starting air regulating valve failed.

Having completed the syntactic and semantic analysis of the sentence, we would recognize *starting air regulating valve* as an example of the prototype *regulating valve*. We would then fetch its description and create an instance of a regulating valve. Next, using the general knowledge about valves (of which regulating valve is a more specific case), and the semantic information about *starting air*, we would modify the just created instance with the fact that the substance the valve regulates is starting air. From the syntactic analysis we would know that *starting air regulating valve* is the subject of the verb *fail*. Using the prototype of the action *fail*, we would create its instance and possibly also further modify the valve instance so that its operational state is recorded. These two instances would now constitute the discourse context so far. Now, suppose the message continues with the sentence:

Unable to consistently start nr 1b turbine.

The processing would be similar to what has been described above for the first sentence. We would create an instance of a gas turbine, would fill its proper name slot with *nr 1b* and finally use the instance as an argument in another instance recording the finding about start problems.

These two sentences come from an actual CASREP. In the starting air system (our initial domain) there are three different valves regulating starting air. Two questions, crucial for understanding the report, might be posed in connection with this short, two-sentence text: (1) which of the three valves was meant in the first sentence? (2) did the reported difficulty with starting the turbine mean that the turbine itself was impaired in some way and should thus be included in the report summary, along with the valve, in the list of damages?

The general knowledge of equipment may tell us a lot about failures, such as: if a machinery element fails, then it is inoperative, or if an element is inoperative, then the element of which it is part is probably inoperative as well, etc. Unfortunately, such knowledge is not enough: there is no way to answer these two questions (not only for an artificial understanding system, but even for us, humans) without access to rather detailed knowledge about how various elements of the given piece of equipment are interconnected and how they work as an ensemble. In our case we could hypothesize (using general knowledge about text structures) that there is a causal relationship between the facts stated in the two sentences. To test this hypothesis, we would have to consider each of the three valves in turn and check how its inoperative state could affect the starting of the specific (i.e. *nr 1b*) turbine. To perform these tests we would need a simulation model. If one of the three valves, when inoperative, would make the turbine starting unreliable, then we could claim that this valve is the proper referent for the *starting*

³ We should point out that in this domain *starting air* is a type of *air*.

air regulating valve mentioned in the first sentence. This finding would allow us to answer question (2) as well. If the failure of the valve is the cause of the trouble with the turbine, it is plausible to assume that, once the valve is replaced, the turbine would work well also. Hence, it shouldn't be reported as damaged.

Another important argument in favor of an equipment model is related to the interpretation of noun phrases (NP). One notable feature of technical texts is the heavy use of nominal compounds. It seems that their average length is proportional to the complexity of the discourse domain. In the domain of the starting air system, examples like *stripped lube oil pump drive gear* are, by no means, seldom occurrences.

The problem with nominal compounds is their ambiguity. The syntactic analysis is of almost no help here. Even using semantic (selectional) constraints, as in [9], substantial ambiguity often remains. When we know that the nominal compounds refer to objects existing in the system, and have access to a model of the system, we can impose much tighter constraints, thus reducing the ambiguity. Model-based causal reasoning (as illustrated above) may reduce this ambiguity further. The problem may be metaphorically described as a jigsaw puzzle: given several pieces (concepts corresponding to individual words from a nominal compound) put them together to build a sensible picture (concept corresponding to the entire nominal compound). The task becomes somewhat easier in cases where we know that nominal compounds refer to concepts existing in the system. In terms of our metaphor this translates into a hint: a set of pictures is given with the assumption that the solution is one of these pictures. Our model plays a role of such a set of aiding pictures.

The above two considerations demonstrate that in cases where the domain is very specialized and complicated (a typical situation for real-life equipment), language understanding systems should not only be provided with general knowledge about the equipment but also have access to its model.

3 Related work

There are two areas of the AI research to which we can relate our work. The simulation model can be considered as an exercise in qualitative physics. The discourse analysis module can be discussed as a contribution to the natural language understanding of narratives.

A good collection of articles on commonsense reasoning about physical systems was published in 1984 as a special issue of *Artificial Intelligence*, reprinted as [4]. The major contributions to this volume, [10], [7], and [13], as well as some other works published elsewhere, like [5], give a good picture of the field. The approach we took for simulation shows some similarities with this research. However, because our main objective is language understanding, we didn't investigate the matter of naive physics at the depth demonstrated in the cited papers. On the other hand, the demands of the discourse analysis lead us to some interesting issues not addressed before. [7] comes closest to our simulation model. We also use a device-centered ontology in which the devices like valves, filters, etc. play the role of the primitives. The *no-function-in-structure* principle, *qualitativeness* (the variables used to describe the behavior of the device can only take on a small predetermined number of values), and the *quasistatic approximation* are central to the PROTEUS simulation. Similarly, in the equipment simulation we rely on three physical constituents: materials (in our case working substances), components (in our case equipment units), and conduits. The major difference is that instead of a sophisticated qualitative calculus (based on *confluences* in [7]), we use a much simpler approach in the form of simple rules, defined for the equipment units, which map inputs to outputs (simulating feedback was not necessary for our purposes). Considering the distinction between structural, behavioral, and functional descriptions [13], we deal with the first two only. Another simplification is our treatment of processes as events; for example, we simulate *corrode* as an event which occurred at a certain time moment.

The most difficult challenge, not present in the research on naive physics, followed from the impossibility of including in the model everything which could possibly be mentioned in the CASREPs (we discuss this in more detail in section 6.4.2). We had to provide a means for modifying the model dynam-

ically (i.e. during text processing), both by adding new components and by imposing new structures on the existing parts. The compositionality principle proved very useful for solving the latter problem. Another interesting issue corresponds to the recognition of objects on the basis of quite long and intricate noun phrases encountered in the reports (see section 6.1). Some parts of the structural and behavioral descriptions had to be specially provided for this purpose. More difficult was the problem of combining both types of information, i.e. that used for recognition and simulation, into one. Still another novelty was the use of generic structures (see section 5.3), which were developed for the purpose of keeping the size of the model as small as possible.

One of the earlier approaches to understanding narratives was based on story grammars [19] which attempted to anticipate general developments common to all stories. They were very limited in their applicability mainly because of their top-down character, which was ill-suited for unexpected input. An effective solution to understanding texts in restricted domains is based on *information formatting*, [20], which uses a classification of the semantic relationships within the domain to derive a tabular representation of the text. Because of the attractiveness of this approach for specialized domains, it was elaborated in [15] for processing CASREPs: a production system was used for interpreting the information in the table created by the information formatting. It was possible thereby to generate short one-line summaries for the reports. However, the system was not well suited to incorporating a rich domain model.

Much recent work on understanding texts stresses the importance of providing a solid basis of pre-stored knowledge necessary to interpret text fragments and to link them into a coherent whole. This knowledge is usually organized in the form of pieces of stereotyped information; the two most widely used mechanisms are *frames* and *scripts*. The approaches based on scripts are based on *conceptual dependency* theory proposed by Schank. One of the characteristic features is their relative neglect of syntactic information. The parsing of a sentence is concept driven, aimed at producing conceptual dependency structures. Scripts account for stereotypical information only. To deal with novel situations the concept of a *plan* was introduced [21] and essentially elaborated in [22]. Plans are primarily used for problem solving in situations where given a goal one has to satisfy it. In understanding, plans are used in a different way: they help to follow narrated goals and actions of participants in the story in order to make inferences essential for understanding. [14] is another example of work in a similar vein.

The research based on frames (e.g. [2]) usually assumed a syntactic analysis (often using ATN parsers) of text as the first stage in the understanding process. Otherwise, the crucial problems were similar: how to organize knowledge (e.g. declarative vs. procedural), how to identify the appropriate frame (script) relevant to the analyzed fragment, what to do in case of wrongly chosen candidates, etc. Because, typically, the attacked stories dealt with human actions, the most difficult problems they raised was the variety of possible plots. This necessitated maintaining rather big collection of frames which added to problems at every stage of processing. Frames were also successfully used for building more specialized understanding systems, like the PAINTING program [6] which was designed for understanding stories about *mundane* painting. One of the assumptions made there was that a program which is expected to understand texts about a certain type of activity should be also able to perform this activity. We embraced this conviction, believing that PROTEUS will be able to understand messages about equipment damages, if it will also have the power of simulating this equipment's behavior. Otherwise, the approaches used for PAINTING and for PROTEUS are quite different. We are not aware of any text understanding system which relies on simulation to a comparable extent. There are similarities with other approaches (e.g. [3]) in the way the structural knowledge is represented: we use a frame-like approach based on the *Symbolics* flavor system.

4 The structure of PROTEUS

Fig. 1 presents a flowchart of how the model is used in the understanding process. [11] describes the overall organization of PROTEUS in some detail.

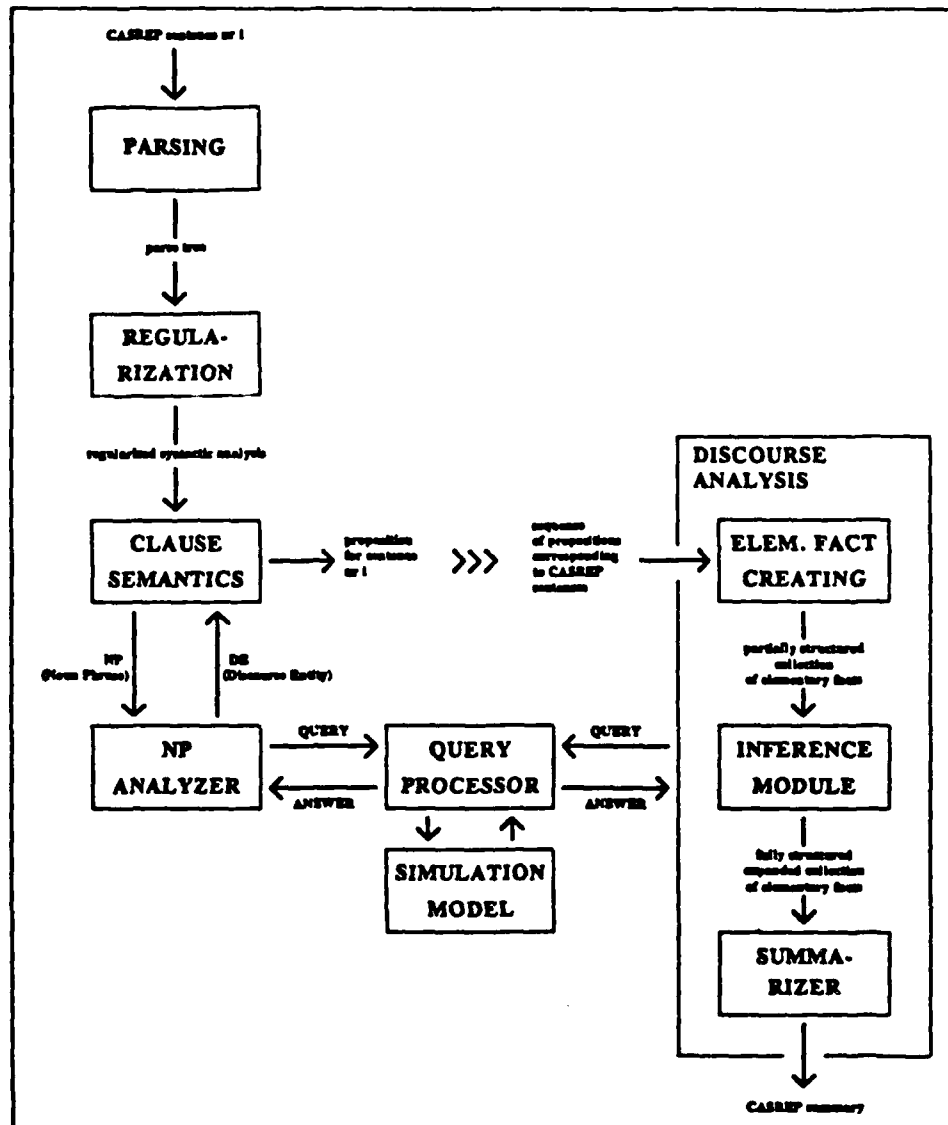


Figure 1: PROTEUS structure.

PROTEUS processes sentences sequentially (first syntax, then semantics, finally discourse). Each sentence is first parsed using an augmented context-free grammar; the parse trees are then regularized, mapping all the different clause structures into a standard verb-argument form. These regularized syntactic structures are then fed to semantic analysis. Each sentence is converted by *Clause Semantics* into an atomic proposition or several propositions connected by logical, modal, epistemic, or temporal operators. An atomic proposition consists of a predicate and arguments which are either discourse

entities determined by the *NP Analyzer* or other propositions.

Once the syntactic and semantic analysis of all the sentences from a CASREP is finished, the created propositions are fed to the *Discourse Analysis Module* as a sequence in the same order as the corresponding sentences in the report. Each proposition is first translated into some number (possibly 0) of *elementary facts*. These elementary facts can be connected with links of various types representing temporal, causal, and other relations. These relations are partially derived from the operators in the representation generated by the *Clause Semantics Module*. Next, the *Inference Module* tries to infer implicit relations, thus augmenting these links, and, if necessary, expanding the collection of elementary facts by inferred and hypothesized ones. This enriched structure of elementary facts constitutes everything PROTEUS was able to understand from the analysed report. Depending on the particular application, a summary is extracted from the output of the *Inference Module*. All parts of the *Discourse Analysis Module* as well as the *NP Analyzer* make use of the information provided by the *Simulation Model* in the form of answers to queries.

5 The PROTEUS Simulation Model

5.1 Characteristics of the domain

The target domains for PROTEUS are *equipment units* (EU): complex technical systems which accomplish physical tasks on demand. These tasks are carried out as serial and parallel combinations of simpler tasks, which are performed by constituent EUs of the main equipment unit. Often these simpler tasks can be decomposed further, leading to a hierarchy of tasks and EUs.

The EUs transmit their effects through various *media*, such as gases, liquids, mechanical movement, and electric current. These media travel from one EU to another through *conduits* appropriate to the different types of media.

5.2 Requirements for the domain model

The simulation model described in this section reflects our assessment of how best to organize the knowledge necessary to solve the language understanding problems encountered in CASREPs. Any knowledge base designed for the purpose of understanding highly technical texts about equipment failures should be able to provide the following kinds of information:

- Detailed *structural information* about the domain equipment. This information is crucial for analyzing NPs which in technical texts tend to be quite intricate. For example, it should provide answers to questions like: is there any pair of referents for the words *adapter* and *hub* which are related to each other by the predicates *adjacent-to* or *part-of*. The ability to answer this question is important for finding the correct referents for the NP *adapter hub*. The structural information is also important for solving certain discourse analysis problems. In some cases, like hypothesizing new facts which would make the reported facts into a coherent whole, it is necessary to find equipment elements which fulfill certain structural conditions (e.g. are two specific gears part of the same transmission chain). In others, like deciding whether a given rule capturing general knowledge about technical equipment is applicable, we would like to know whether a particular equipment element has features required by the rule.
- *Functional information* about the domain equipment. This is the information about the role which individual parts play when the equipment is operating normally. Its significance parallels that of the structural information, i.e. it is essential for analyzing NPs, hypothesizing new facts and checking equipment element features. Consider for example the NP *starting air valve*. To determine its referents it is necessary to be able to decide for every valve in the equipment whether the air which can flow through it serves as a starting medium for any other equipment element. In

case of pump drive gear we would like to be able to determine which of the gears in the domain equipment drives a pump.

- *Operational information* about the domain equipment. This is the information about the states of particular parts in certain situations, like when some other part is damaged and still another part is in a certain working state, for example engaged. This kind of information is essential for finding causal relationships between reported facts, e.g. can the damage of a particular pump be the reason for an alarm. The operational information is also used to determine prerequisites for given states, like what state the diesel must be in when the SAC is engaged; the knowledge about prerequisites allows to solve some time problems, e.g. determining partial ordering between start and end nodes of intervals during which certain states are true.

Besides the information directly related to problems of language understanding we were also concerned about capturing in the knowledge base the information necessary for graphical display. This followed from our conviction that natural language understanding systems in technical domains should be equipped with well designed interactive graphical interfaces.

Having considered several different forms for a knowledge base which would meet the needs described above, we concluded that a simulation model for the domain equipment would be the best way to combine the various kinds of information. The advantage of this approach over a rule based model is its compactness (from a simulation model one can derive a very large number of rules) and its generality (it is easier to interpret phenomena newly encountered in the reports in the context of a simulation model than of a rule system).

It should be noted that not all information necessary for understanding the reports comes from the simulation model. Certain types of knowledge, e.g. the general knowledge about equipment, is best expressed in other forms (e.g. production-like rules), and we make use of it occasionally. Relatively speaking, however, it is of secondary importance and thus can be neglected in the following description.

5.3 The structure of the simulation model

PROTEUS models have the structure of recursive transition networks. They consist of *nodes* connected by directed *links*. The nodes correspond to the constituent EUs of the system; the links to the conduits connecting the EUs. The hierarchical structure of the EUs is reflected in the hierarchical structure of the networks. To represent the internal structure of an EU, we have the corresponding node point to another network in the model. Such (non-terminal) nodes are called *system nodes*. This recursive refinement must stop somewhere; the (terminal) nodes — which don't have underlying networks in a given model — are called *simple nodes*. We shall use the term *model-networks* to refer to the entire collection of networks modelling a piece of equipment.

Associated with each link is a *working-substance* (WS). These WSs correspond to the media entering and leaving an EU (for example, the rotary motion provided to a pump and the fluid entering and leaving the pump). We can think of the WSs associated with links entering and leaving a node as the input and output data of the node.

The nodes, links, and WSs are represented as sets of (attribute, value) pairs. These pairs are used not only to define the system structure, but also to express all other, non-structural information. For nodes and links we call the attributes *roles*; for WSs, we call them *properties*. The attribute values may be pointers to other nodes, links, or WSs, numbers, symbols, or propositions. Some of the values are time-dependent reflecting the particular state of the model in the course of a simulation.

Each node has four (time-independent) roles carrying structural information: *part-of* (a pointer up in the hierarchy of networks), *structure* (a pointer down in the hierarchy of networks), *to-links*, and *from-links* (see below for more about links). The current state of a node is recorded in the (time-dependent) roles *control-state* (which reflects the position of some control switches) and *readiness-state* (which describes the possible damages reported in CASREPs). These two roles are important for reference and simulation but have no direct impact on the graphics. Some nodes also have (time-independent)

roles for simulation parameters, such as **transmission-ratio** for nodes describing couplings. There are also roles used exclusively for graphics, such as: (time-independent) **size** and **grid-location** specifying for EUs' icons their sizes and places on the screen and (time-dependent) **moon-position** for nodes describing spin elements. This last role keeps the current angular position of the small globe which revolves around the node's icon on the screen to represent rotary movement.

There are two types of links in model-networks: *simple links* which connect two nodes in the same network, and *ancestor links* which incide on a node from a higher level network or exit a node to a higher level network. The two main functions links play in the model are: to complement nodes in establishing networks and to relate WSs (whose parameters are often mentioned in the reports) to EUs (which also are commonly referred to in the reports) — WSs are associated with links which are in turn connected to nodes describing EUs.

Each link has the (time-independent) roles: **ws-ptr** (working substance associated with the link) and **to-** and **from-end** (link's end and start nodes). The time-dependent aspects of links are important for display purposes only and are derivatives of time-dependent attributes of the WS associated with them. The graphical display for links is much more complicated than that for nodes because links are the main vehicle of the dynamic graphical interface of PROTEUS: link icons are animated, reflecting such features of the working substances as flow and pressure in the case of liquids and gases or speed in the case of rotation.

Each WS has a (time-independent) attribute **link-ptr** which takes as value the set of all the links which point to the WS. The time-dependent properties of WSs used for the simulation vary from case to case depending on media: for liquids they may be: **pressure**, **temperature**, **flow**, **color**, and **purity**; for rotary movements: **rotary speed**. Usually, some of these properties have their correspondents used for reference purposes. For example, for liquids we may have **normal-operational-pressure** or **normal-operational-temperature**. This is necessary because otherwise it would be impossible to find a correct referent for a noun phrase like *high pressure oil pump* in a state of the model in which the pump is not working. Modifiers of this kind usually refer to a situation of normal operation of the equipment: independent of whether this pump is running or not or whether — because of a failure — the oil it discharges has only low pressure (this low value of the pressure is recorded as the property **pressure**, whereas the property **normal-operational-pressure** remains unchanged, i.e. has a high value), it can always be referred to in the reports as a *high pressure oil pump*. For a more detailed discussion of this example see the section on noun phrase analysis.

Usually, a model network is subsumed by only one system node. However, in order to avoid unnecessary escalation of a model's size, we allow a system network to be subsumed by several system nodes. We call such networks *generic*. This mechanism is useful for EUs some of whose components are identical. For example, in our domain of the starting air system, there are three identical air compressors each of which requires over 100 nodes. Generic networks allow us to avoid this costly repetition by modelling an air compressor only once and for each of the compressors having a node whose **structure** attribute takes the same value, namely the pointer to the generic network. Two modifications are necessary in order to make generic networks work. Firstly, the **to-ends** of a generic network's exiting ancestor links and **from-ends** of a generic network's entering ancestor links can no longer be individual nodes on a higher level; for each ancestor link in a generic system there will be more than one way of ascending to a higher level (either forwards using the **to-ends** or backwards using the **from-ends**). In fact there are as many different ways to do this as there are system nodes subsuming the generic network. Secondly, all the time-dependent attributes of the nodes, links, and WSs of a generic network will have distinct values for each system node subsuming the generic network. One straightforward solution to these problems, taken in PROTEUS, is to use as values of the attributes mentioned above, arrays of size equal to the number of system nodes subsuming a given generic network. With this approach an EU which is modelled by a generic network is fully described by this network and an array index.

5.4 The simulation process

As we indicated at the beginning of this section, one of the main purposes for which we intended our model is the qualitative simulation of the modelled EUs (in most cases a precise quantitative simulation is not required for language understanding). Simulation is performed by an event-driven algorithm which is triggered by an external event and continues until a stable state is reached. The model handles three types of external events: (1) operator-initiated actions, (2) undesired accidental events corresponding to reported failures and damages, (3) so-called *demons* (cf. below), i.e. events resulting from the automatic behavior of the equipment. Although it is possible to have several levels of detail in the model, the simulation is always conducted on the lowest level, i.e. on the level of simple nodes. To each of these nodes a *model function* is assigned. A model function of a node consists of two parts:

1. for any modelled change in the node's readiness state, it defines the changes to WSs associated with its input and output links;
2. for any modelled qualitative change in any of the WSs associated with the node's input and output links and any values of the readiness and operational state attributes, it defines the changes to the WSs corresponding to all other input and output links.

The simulation can be viewed as an interweaving of intervals of stable and unstable states of the model. An unstable phase is started by an external event such as a control action or a damage at some node of the model. Once it happens, the model function of the node is applied and all changes in the WSs associated with the node's input and output links are determined. These changes propagate (forwards through output links and backwards through input links) to the neighboring nodes. Each of the nodes affected in this way is considered in turn and its model function determines how the changes propagate further. This process continues until there are no more changes. At this point simulation enters its stable phase and remains in this state until another external event occurs.

One feature of model functions is that there are no delays in propagating the changes triggered by an external event. However, some of the analyzed messages mention (explicitly or implicitly) such delays, e.g. the interval between the act of starting a turbine by an operator and the time when the turbine gets actually started may extend up to several minutes. To accomodate such cases we introduced *demons* as a third kind of event starting an unstable stage of the simulation. For demons to work a simulation clock is required. A demon consists of two parts: a time point when it should be activated and a demon procedure name. Demons are set during the execution of model functions of nodes. The demon procedure is executed when the clock time reaches the demon activation time. It determines what changes should be made to the WSs at the input and output links and possibly to the operational state of the node. The propagation of these changes proceeds like the one described above for control activities and damages.

None of the messages we analyzed required simulation of feedback. Therefore, the PROTEUS simulation was not designed to deal with feedback. Considering the possibilities of using PROTEUS-like models for other applications than language understanding, notably expert and tutorial systems, this must be certainly viewed as an important limitation. The research on *incremental qualitative simulation* [7] gives us reason to believe that extending the coverage to feedback is possible in the framework of our approach to simulation.

5.5 Generality of the model

We tried to design PROTEUS in such a way that it may be adapted easily to new equipment. Clearly, the model has to be built anew each time we want to use PROTEUS for a new equipment unit. To minimize the work required, we distinguish between knowledge about equipment in general and knowledge about a specific piece of equipment. We tried to achieve this duality of knowledge using prototypes and their instances: the model would be built of instances of prototypes. The prototypes constitute part of the

general knowledge base. In the instances we store only the information which is specific to the object described by the instance. For example, in the case of a gearbox, the information about its function (speed change) should be stored in the prototype, and only the ratio of this change should reside in the instance of a specific gearbox. Also the information about how a specific gearbox is used in the domain equipment must be kept in the instance. Of course, the prototype-instance scheme ensures that all the general knowledge connected with the prototype is also accessible from instances of this prototype. We found the rich repertoire of programming tools constituting the flavor system in *Symbolics-Lisp* a very convenient vehicle for implementing this strategy.

5.6 Level of detail

In the preceding section we have provided a mechanism for a stepwise recursive refinement of the model. The EUs we want to model are both big and complex. For example, our initial domain, the starting air system, is described in the ship's manual on 28 pages of text, figures and tables. We need some kind of simplification. A possible approach would be to refine the hierarchy far enough so that everything which potentially may be referred to in the reports would have a description in the model. This, however, seems impractical. Consider a typical sentence from one of the reports: *Investigation revealed a broken tooth on the hub ring gear*. Considering that there are several different gears in our starting air system and each of them has many teeth which are very much alike, it's obvious that creating a separate description for each of them wouldn't be reasonable. The same remark is true for balls in bearings or for connecting elements like screws, bolts or pins. On the other hand, information about the tooth conveyed in the above sentence cannot go unnoticed. The solution we adopted for such elements is not to include their descriptions in the model on a permanent basis but to keep open the possibility to create and add them to the model if such a need arises during the analysis. A rule of thumb for deciding whether a particular element deserves a permanent place in the model can be formulated as a question: How much information specific to this element is necessary to solve understanding problems, like finding referents (see the section on nominal compounds) or making inferences? As an example of the latter, let's consider a specific gear. We would like to know, among other things, the gear's role and place in the modelled equipment so that, in case of its damage, we could determine the impact on the equipment. Information of this type can be deduced neither from the analyzed text nor from general knowledge about gears. It must be known in advance. Our way to achieve this is to keep the gear's description permanently in the model.

There are, however, elements like teeth which have so little relevant structure that they are always referred to as *tooth*, *teeth* together with the element higher up in the part/whole hierarchy (let's call such an element a *host*). Thus, it is not necessary to maintain any specific information about them in the model. It will suffice, if we are able to create their descriptions when they occur in the text. All the possible information we will ever need to include in such descriptions will come from the text. The information relating such elements with other parts of the equipment will come from their hosts. For example, the impact of a tooth's damage on the equipment may be derived from the functional information connected with its host. We will return to this issue of statically and dynamically modelled objects later in this paper.

It is important to notice that there is nothing absolute in distinctions such as the one made above. It is conceivable to have a piece of equipment of a larger scale than the SAC, where elements like gears are not essential enough for us to be bothered with their shapes or locations; if broken they probably would be referred to by giving the higher-level element of which they are part. In such cases we would rather treat gears like we treat teeth here.

For simulation alone, it would be sufficient to keep in the model only those networks which are on the deepest level. However, because the texts refer to equipment parts at various levels of detail, we have to maintain all intermediary levels of the model. Furthermore, because of the requirement of relating facts about such parts by means of causal links, we should also be able to cross levels in network traversals. Such traversals are possible using the ancestor links.

5.7 An example: the starting air system model

As our initial domain we have chosen the "starting air system" used for starting gas turbines on Navy ships. The model consists of 15 networks with a total of about 175 nodes.

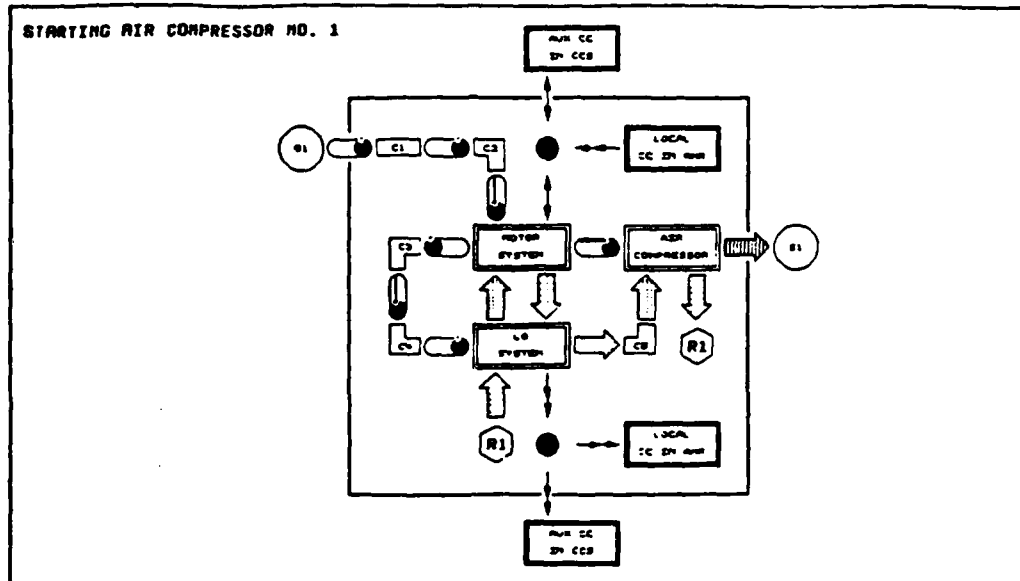


Figure 2: Model network for SAC.

We show here two networks and one control console of the model. Both figures are *Symbolics* screen images generated by PROTEUS from the model networks. Fig. 2 is the starting air compressor (SAC) network. The rectangles enclosed in double lines are the system nodes, which point to more detailed networks in the model. One of these is the LO (lube oil) network, shown in Fig. 3. Simple nodes are represented by icons, such as the filter, cooler, and valve in Fig. 3. The rectangles enclosed in heavy black lines represent control consoles from which the system may be operated and monitored. One of these is shown expanded in Fig. 3. Clicking the mouse on the ON or OFF buttons turns on or off the corresponding EUs. The control consoles also have indicator and alarm lamps as well as instrument displays. The arrows represent links associated with the flow of WSs like gases and liquids. The insides of these arrows show some of the parameters of these WSs. For examples, the bars indicate air and the dots oil. The density of these bars or dots corresponds to the pressure of air or oil, respectively. When the bars or dots remain stationary, it means there is no flow in the conduit corresponding to the link. The rods represent links associated with rotation WSs. The rotation is indicated by small filled-in globes (moons) on both ends of a rod and a line connecting these globes. The speed with which these globes rotate indicates the speed of the corresponding WS. These dynamic displays provide a direct visual presentation of the system's understanding of a message (oil may stop flowing or a gear rotating). They are achieved as a side effect of the simulation used for understanding purposes. The links crossing the outline boundaries are ancestor links (links which connect the network to other parts of the model);

the other arrows are simple links.

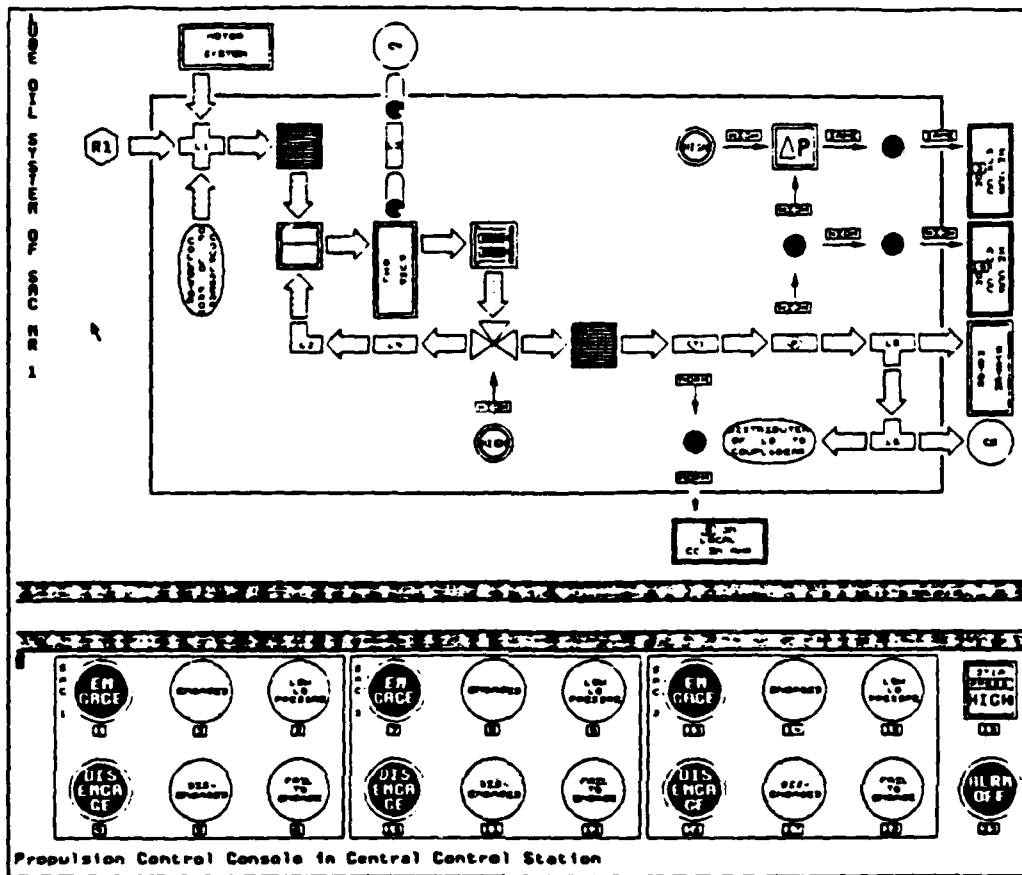


Figure 3: Model network for SAC LO System.

6 Noun Phrase Analysis

6.1 The role of noun phrase analysis

The goal of the *noun phrase analyzer* (NPA) is to convert a noun phrase into a set of referents that may be used by subsequent stages of the system. For example, for a noun phrase describing an EU, the NPA would return a pointer to the corresponding node in the equipment model.

Why is this a difficult task? As we noted earlier, the texts in this domain are replete with long compound nominals, such as *stripped lube oil pump drive gear*. Parts are not always referenced by the same name, even within a single manual (for example, *surge control system* and *compressor surge valve assembly* refer to the same EU); parts may be described by position, function, or other properties. Consequently, we cannot simply use a large term dictionary; we must analyze the structure of each noun phrase. Specifically, we must determine what modifies what within the noun phrase; for example, that *lube* (short for *lubricating*) modifies *oil* rather than *pump*, *drive*, or *gear*. We must also identify implicit relations among elements; for example, that *gear* is an adjacent part to *pump*. Syntax is of little help

in this task, so the PROTEUS syntactic analyzer passes prenominal modifiers through as unanalyzed strings. It is thus left to the NPA to determine the structure and implicit relations of the noun phrase.

The equipment model serves two functions for the NPA: First, the model is used to confirm possible relations between noun phrase constituents (for example, that there is a gear which is adjacent to a pump). Second, once a structure has been determined for the noun phrase, the model can be used to provide a set of referents for the phrase. We call these referents *discourse entities* (DEs).

The interaction between the NPA and the model is mediated by the *Model Query Processor* (MQP). The MQP evaluates domain predicates relative to the equipment model, and creates DEs when needed. In this way the NPA need deal with only two levels of representation, the linguistic representation and the representation in terms of domain predicates, and is made independent of the details of the equipment model representation.

6.2 Word classes

For the purposes of the NPA, the words which may occur in noun phrases are grouped into *semantic classes*. These classes reflect the ways in which the words may combine to form larger phrases. They are specific to the equipment domain but are independent of the specific model representation. Each noun phrase (and each constituent phrase within a noun phrase which is identified by the NPA) must be assigned an internal representation for use in subsequent processing; this representation is the *discourse entity* mentioned above. Different types of entities require different internal representations, so the entities are grouped into *discourse entity classes*. These classes are described below (cf. section 6.4).

Words which cannot stand alone as noun phrases, but only participate in noun phrases as names of relations (e.g., *regulating*) or as arguments of relations (e.g., adjectives of shape) will not have corresponding DEs. Thus these words will belong to a semantic class but not a DE class.

6.3 The Analysis Procedure

The first stage in the analysis consists of attempts to impose all meaningful structures on the words which are constituents of the NP. For this purpose we use the semantic classes of the words and try to combine them using the predicates. The set of predicates we use for this purpose is a result of a detailed manual analysis of part of our corpus. We started by identifying three groups of predicates:

1. structural predicates: Adjacent-To, Assemble, Couple, Location, Made-Of, Part-Of, Shape;
2. functional predicates: Drive, Lubricate, Operate-On, Regulate, Start;
3. special predicates: Alarm, Measure, Name.

We considered each of these predicates in turn and tried to decide which semantic classes should best describe the predicate's arguments. In doing this we tried to come up with the broadest possible classes within the limits of the technical sublanguage. Some examples of these semantic classes are:

equipment substance feature value measurement-unit shape direction

The predicates together with the appropriate semantic classes constitute patterns of the form

(Pred arg₁ arg₂ ...)

where Pred is a predicate of the domain and the arg_i are semantic classes. At the current stage of development we have 28 such patterns.

The NPA begins by fetching (from the dictionary) the semantic class of each word in the noun phrase. Constituents are combined bottom-up based on their semantic classes and the set of patterns

described above. Most current systems validate the application of such patterns through selectional checks (constraints on the argument classes of each predicate) ([9]). We perform such checks and then go a step further and check for the existence of the specific relation between specific entities. This is done through a call to the MQP. The MQP is first invoked for each word to obtain its internal representation in the model, and then for each proposed predicate to verify the existence of the corresponding relation in the model. If the relation exists, the MQP returns the representation for the head constituent of the relation.

Applying these rules to post-nominal modifiers is relatively straightforward: the predicate can generally be determined from the preposition, and the arguments of the preposition are explicit and delimited. After processing the post-nominal modifiers, the NPA turns to the pre-nominal modifiers. Analysis of these is considerably more difficult. The problem is to decide what predicates should be used and what are the arguments of these predicates. Both the predicates as well as their arguments may be given explicitly or implicitly. Examples are: *temperature regulating valve* (the predicate and both its arguments are explicit), *drive gear* (the predicate and one of its arguments are explicit, the other argument, the object of *drive*, is implicit), *pump shaft* (the predicate, *part-of*, is implicit, both of its arguments are explicit). The NPA considers the semantic features of the items, together with order constraints, to match the items with arguments of some canonical predicate. A match is considered successful if it is possible to identify some (not necessarily all) of the arguments of the predicate among the modifiers. For verification purposes, it is assumed that the empty arguments match anything. Once a matching canonical predicate has been found and as many of its arguments matched with the modifiers as possible, the NPA may pose a verification query to the model.

In some cases, the proper argument to a predicate is an object or concept related to the explicit object. For example, in *air regulating valve* the object of *regulate* is not air itself, but one of the properties of air. A similar case is *high pressure pump*; here, the problem is to find a relation between an EU (the pump) and a medium property (high pressure). To account for such cases, if none of the canonical predicates matches the modifying and the modified items, the NPA repeats the matching procedure with objects or concepts which are variations of these items. Possible variations are: generalizations, fragmentations, specializations (properties for WSs, roles for EUs) of the item referents. For example, for the phrase *high pressure pump*, if we fail to find a predicate which relates a pump and a property of a medium, we next try with a pump and a medium which has that property.

6.4 Model Query Processor

The MQP serves both the NPA and the Discourse Analyzer. We will discuss here the queries used by the NPA, i.e. requests for the DE for a word, and predicate verification queries; the queries used by the discourse analyzer are discussed in section 7.5.

6.4.1 Discourse Entity Classes

Noun phrases refer to different types of concepts and so must have different types of internal representation. We have grouped the NPs of this domain into classes on the basis of the form of the DEs we create for them. For some classes the DEs are constituents of the model previously described; for others we dynamically create structures analogous to those in the model. The *discourse entity classes* are:

1. **Equipment Units (EUs).** Used as modifiers (e.g. *compressor* in *compressor shaft*) or as main objects (e.g. *shaft* in *compressor shaft sheared*). Their DEs are nodes in *model-networks*. Most of these nodes exist in the model permanently. Some, however, are created during analysis of the NP (see discussion below).
2. **Media.** Used as modifiers (e.g. *starting air* in *starting air compressor*) or as main objects (e.g. *lube oil* in *lube oil contaminated*). Their DEs are *working substances* (WSs). All media mentioned in the

reports, except those used as modifiers of non-permanent EUs, are assumed to have corresponding WSs in the model.

3. **Stuffs.** Used as main objects only. Their DEs are created when a NP describing a stuff is encountered in the text. Their DEs consist of three parts: (A) material, (B) form, (C) quantity. Examples are: *non-metallic particles, chips, metal chunks*.
4. **Media Properties.** They are used as modifiers of media (e.g. *high pressure* in *high pressure lube oil*), or represent concepts on their own (e.g. *LO temperature* in *LO temperature increased sharply*). Their DEs generally consist of three parts: (A) a WS, (B) name of the property, (C) property value. Not all of these three parts are referred to explicitly in a NP. Possible combinations are: [A, B, C] (e.g. *low output air pressure*), [A, B] (e.g. *LO pressure*), [B, C] (e.g. *high speed*), [B] (e.g. *temperature*). A new DE is created each time a property is mentioned in the text.
5. **Media Property Values.** They are used as parts of 4, above (e.g. *normal* in *normal temperature*), or represent concept on their own (e.g. *zero* in *lo pressure dropped to zero*). They may be numbers (e.g. *zero*), strings (e.g. *nr 1B*), or predicates (e.g. *above 65 psig*). A new DE is created each time a value is mentioned in the text.

6.4.2 Creating DEs for Words

One issue we have already addressed several times is the mixed static/dynamic characteristic of our representation. The main equipment structure is recorded statically in *model-networks*, while DEs for low-level EUs and words of certain other DE classes are created dynamically. Since the other modules are designed to be independent of the model representation (in particular, the static/dynamic distinction), this distinction must be hidden by the MQP.

We use four different approaches to the DE request queries. The first two are exemplified by the class EU. The third one is illustrated by classes Stuffs and Media Properties and the last one by the class Media.

For permanently modelled EUs, the dictionary contains a set of pointers to all nodes in *model-networks* which are its DEs. Such a set assigned to a word is returned if the DE of the word is requested. For example, for the word *pump* the set of pointers to all pumps in the model is returned. The approach becomes more complicated in case of DE requests for words which correspond to EUs which are not premodelled. We have two kinds of such EUs:

- EUs as parts of other EUs whose DEs are in the model. As indicated in the preceding section dealing with the model, it is impractical to represent every tiny element of the modelled equipment. However, such elements occasionally break and are thus mentioned in the casualty reports. For any such EU we create a new node. The pointer to this node is returned as a result for the DE request. In addition, we must establish a connection between this new node and the rest of the model. This is simple, because the only connection of such a node with the model is through the *part-of* relation with some node in *model-networks* which is mentioned (explicitly or implicitly) in the text. An example here is: *connecting pin in pump drive assembly*; the connecting pin is not statically modelled.
- An EU as a superstructure of some subset of EUs which have been modelled, but not grouped together in a single network subsumed by some node which could be the DE for the EU. The reason for not having such EUs modelled *a priori* goes back to the practicality considerations again: the number of different ways in which EUs may be grouped together in a single description is simply too large to be represented explicitly in the model. An example here is: *the coupling from diesel to SAC lube oil pump*. All the spin elements which constitute this coupling have been modelled but are not in a single network subsumed by a node which could have been the DE of this noun phrase. However, for inference purposes and for reference resolution in later parts of the text, this

coupling must be identified as a node in the model. A node of this kind has to be created and returned as a result for the DE request. In contrast to the above case, establishing a connection between the new node and *model-networks* is a complex task involving several input and output links.

Each word of the classes *Stuffs* and *Media Properties* has assigned in the dictionary a prototype of a corresponding DE which is instantiated when the DE for a word is requested. The pointer to this newly created DE is returned. Words of the class *Media* could be treated like EUs: some of the WSs exist in the model permanently while others have to be created when they occur in the text. We decided not to take this approach here for two reasons: The number of pointers returned would be relatively large (in case of air, for example, it would be almost equal to the number of air links in *model-networks*). Secondly, in most cases, the modifiers are of no great help in reducing this number. More often, the context and the default values help to disambiguate the reference. Hence, for words of this class we create descriptions in the same manner as for words of classes *Stuffs* and *Media Properties*. However, when such a DE is about to be interpreted as part of a predicate, we use it and the context information to find the correct referent in the model. Consider, for example, the sentence *Air pressure increased sharply*. Analyzing the NP alone, it is not possible to figure out which particular air WS is meant. However, the information about the context allows us to make a reasonable assumption that the right referent is the air WS which is associated with the link entering a pressure measuring instrument.

6.4.3 Predicate verification queries

The *predicate verification queries* have the form:

```
(MQ-Pred-Verif predicate head
      arg1name arg1val
      arg2name arg2val
      ...)
```

where *predicate* is one of the predicates listed in section 6.3, *head* must be one of the *arg1name*, and *arg1val* are DEs returned as result of the DE request queries described above or of predicate verification queries described below. *arg1val* are either sets of DEs or individual DEs. Named rather than positional arguments are used because the queries may be invoked with only some of the *MQ-Pred-Verif* arguments. The MQP provides defaults for all the missing arguments. The *head* indicates which one of the arguments should be modified to be later returned as the query result. We call the *argval* of such argument the *head argument*. Depending on whether the head argument consists of premodelled DEs or of DEs dynamically created as a result of queries to the MQP, two cases are possible:

- The head argument consists of premodelled DEs. We treat an individual DE as a one-element set of DEs. The query is then understood as a filter condition. The MQP considers all elements of the head argument set one by one, checking which ones pass the test described by the query predicate and collecting them to form the result to be returned. If any of the other arguments are sets as well, the test succeeds if any combination of members from these argument sets makes the predicate true. If none of the members of the head argument set pass the test, the MQP returns *nil*, and consequently the NPA must drop this attempted analysis of the NP from further consideration. For example, for the query

```
(MQ-Pred-Verif Shape :equip-unit
      :shape ring
      :equip-unit (gear1 gear2 gear3))
```

assuming that only *gear₂* and *gear₃* are ring gears, the MPQ returns the set (*gear₂ gear₃*). If none of the three gears were a ring gear, the returned result would be *nil*, and the NPA would

have to try (if possible, e.g. in case the NP were *ring gear assembly* referring to an assembly of gears arranged in a snape of a ring) to consider *ring* as a modifier of some other NP constituent.

- The head argument consists of dynamically created DEs (either just one or a set). In this case the query is understood as an additional piece of information about the objects described by the head argument. The MQP updates the DEs of the head argument in such a way that this information is accounted for in the returned DEs. This may be done either by filling some roles in the head argument DEs, or by creating modified copies of them. An example of the former is the query

```
(MQ-Pred-Verif Lubricate :ws
                  :ws wdesc7)
```

which is posed while processing the NP *lube oil*. The returned result is the *wdesc7* with its function role filled with (Lubricate *any*). For a slightly different NP *compressor lube oil* the query would be

```
(MQ-Pred-Verif Lubricate :ws
                  :ws wdesc7
                  :equip-unit (compr1 compr2 compr3))
```

Here, the MQP would return the set (*wdesc7₁ wdesc7₂ wdesc7₃*), where *wdesc7₁* would be modified copies of *wdesc7*; their function roles would be filled with (Lubricate *compressor₁*).

The evaluation of MQ predicates varies from case to case. It is straightforward for (MQ-Pred-Verif Made-Of :object :material), where only :object may be specified as a head (*metal particles*), and where it is sufficient to compare the role filler of the tested object with the other argument's value. It is more complicated for (MQ-Pred-Verif Operate-On :medium :object). Here, both arguments may be used as a head (*oil pump* vs. *SAC oil*). Furthermore, when the object is a system node, it is necessary not only to examine its input and output links, but also — if there is no WS associated with any of these links which has the medium as part of its description — recursively analyse all the nodes beneath the one corresponding to the object.

The evaluation of the MQ predicates may sometimes involve a quite complex analysis of the model. An example of this is (MQ-Pred-Verif Drive :driving-object :driven-object). Not only may both arguments be heads (*pump drive gear*), but because in most cases only one of them is specified explicitly (e.g. *driving adapter hub*, *driven gear*), the default for the missing argument must be determined. The evaluation procedure, in contrast to the previous case, is different depending on which argument has been declared head. Let's consider the case when the :driving-object is head. This kind of modifier is used in our domain for EUs which generate or transmit a rotary movement. These EUs are arranged in chains of couplings starting at the source of a rotation and ending at places where the rotation is converted into some other type of WS. In every coupling from this chain we could distinguish between a driving and driven element. However, in our corpus the only cases where this modifier has been used are:

1. elements which are on boundaries of higher level system nodes, for example, an adapter hub (lowest level) connecting a diesel with a transmission system (both higher level system nodes);
2. elements which start chains of couplings in networks subsumed by system nodes, for example a shaft which starts a transmission system;
3. special arrangements, for example, *driving* and *driven* gears in a pump.

Cases from groups (1) and (2) are resolved by analyzing the structure of the model, whereas cases from group (3) are solved by examining function slot fillers of the argument DEs.

6.5 Reference Resolution

If at the end of NP analysis we are left with a set of more than one DE, we have to disambiguate the reference. The reference resolution procedure has four parts. The first part of reference resolution is invoked as soon as the NPA finishes and consists of the following steps (we call the DEs to be disambiguated *candidates*; they are of a certain type, such as *diesel* or *shaft*):

1. if any of the candidates is on the focus list (this is a list of the DEs corresponding to all the NPs encountered so far in the analysis of the message) choose this candidate as the intended referent;
2. otherwise, if the candidates' type is listed on the default table, choose the default DE as the intended referent;
3. otherwise, mark the ambiguity for later resolution.

This approach follows [16].

The second attempt to resolve disambiguity is made by the clause semantics. The approach here uses two source of information which can assist disambiguation:

1. the semantic constraints imposed on arguments of clause predicates;
2. specific knowledge about the structure of domain equipment.

Case (2) was illustrated in the preceding section where we have shown how to disambiguate references to *air pressure* (air pressure at any point in the equipment is a possible referent here) in the context of *Air pressure increased sharply*. If clause semantics fails to resolve the ambiguity, discourse analysis can be of help as illustrated by the example of *starting air regulating valve* discussed in section 2, above. As a last resort PROTEUS engages the user in an interactive session: an explanatory text is shown in the notification window, the nodes corresponding to the candidate DEs are highlighted on the display one by one, and the user is expected to select one of them with the mouse. The DE corresponding to the chosen node becomes the selected referent.

7 Discourse Analysis

7.1 How much understanding do we need?

One basic objective in understanding the messages is determining the *cause* of the malfunction. In some cases this information is not explicitly provided. Even reports which have explicit information on equipment damage or malfunction can pose problems. We give two examples:

1. Several problems might be reported which are causally related to each other, so that fixing the first problem from such a chain is enough. Consider for example the pair of sentences: *Starting air regulating valve failed. Unable to consistently start nr 1B turbine*. The malfunction of the 1B turbine shouldn't lead to the conclusion that the turbine is broken and therefore needs repair or replacement. One should recognize that, once the starting air regulating valve has failed, it is not possible to start the turbine, so probably only the valve has to be fixed.
2. A report may mention a problem which has been fixed by the crew, so no intervention from outside is necessary, for example, *Input drive shaft found to be seized.⁴ S/F [ship's force] reinstalled old SAC utilizing new drive shaft*. These examples indicate the importance of *causal* and *temporal* relationships in understanding some messages.

⁴This is an inference from a sentence in the CASREP, not an actual sentence.

Another basic objective is understanding the *effect* of the malfunction. This information is crucial for maintaining an accurate picture of the readiness of the ships from which the CASREPs come. It is desirable here to recognize the highest level on which the functionality has been affected. For example, if a CASREP mentions both a problem with engaging a compressor as well as a problem with starting a turbine, it should be possible to recognize that the latter fact gives a more accurate picture of the readiness of the ship.

This discussion has concerned equipment functionality which has been mentioned *explicitly* in the reports. However, if we broadened the coverage of the model and differentiate among types of damage, it should be possible to infer the effects of the malfunction even in cases where it is not given explicitly.

The primary data structure for this understanding process is a collection of *elementary facts* linked by causal and temporal relations. This representation is discussed in detail in the next subsection. This structure is built in two phases: the first phase analyzes the information explicitly in the message. The second phase uses model-based causal reasoning to augment the causal and temporal links. The following two subsections examine these phases. Appendix B provides a detailed example of the procedures described in this section.

7.2 The Structure of the Discourse Analysis Results

The structure of the results produced by PROTEUS for a CASREP should be flexible enough to capture all the essential information such as was described in the preceding section. Each CASREP is processed by various Navy organizations from different perspectives. It is therefore natural that the aspects of a CASREP which each place is primarily interested in, differs from case to case. In order to be able to tailor the results to these special needs and at the same time to maintain the highest possible uniformity of the Discourse Analysis Modules across the distributed systems, we decided to produce the results in two stages. We first create a collection of *elementary facts* onto which a structure of causal and temporal relationships is imposed. These results are application-independent and contain everything PROTEUS was able to understand. We then build a *summary* which meets specific needs. The summary is created by extracting from the collection only those facts which are relevant for a given application. We will discuss summaries in more detail below. A user who wants to look beyond the information provided by the summary may access additional information from the collection. PROTEUS provides for this purpose a menu-driven interpreter of queries allowing access to more detailed information about the analyzed report.

Sentences in the reports describe a whole range of different phenomena related to the domain equipment, such as damage, functionality problems, corrective actions, behavior scenarios, etc. From the viewpoint of an inference system it is desirable to represent these phenomena using a few basic (canonical) elements for which basic inference operations could then be defined. We call these basic elements *Elementary Facts* (EF). The main vehicles for making inferences are the simulation in the model and the general knowledge about the domain equipment. Therefore, only facts about the equipment parts and the working substances on which they operate are used in the reasoning process. For example, if a report would indicate operator inebriation as a cause of certain damage to the equipment, we wouldn't be able to capture this causal relationship in the model, nor would we find an appropriate rule in the general knowledge data base. Furthermore, only selected aspects of the static and dynamic states of EUs and WSs are of interest to the inference module. For example, information about the sender of a particular replacement EU has no place in our representation. These decisions about what to include in the representation were based on our assessment of the needs of discourse analysis for our corpus of CASREPs.

One important feature of the EFs is their relationship to the simulation in the model. Only certain facts can be *simulated directly* and *independently* of other parts of the model. Among these are the control procedures for activating, deactivating, and setting EUs, which are initiated (both in reality and in the simulation) by manipulating control console buttons and switches. These buttons and switches are the external system inputs. Facts describing states resulting from damage are also simulated directly.

We do not simulate the processes which lead to damage, but rather simulate directly the final damaged states of EUs by changing their descriptions in the model.

Another category of facts are those which describe states which are *dependent* on activities elsewhere in the simulated equipment. For example, low lubricating oil pressure can be the consequence of many different scenarios, such as damage to the oil pump coupling. Setting this oil to the indicated pressure in the simulation model and disregarding the rest of the model would produce an inconsistent state of the model, and so is not allowed. Therefore facts of this type can be simulated indirectly only, i.e. by simulating specific facts from the first (directly simulatable) group.

A third category of facts describe transitions between stable states of the equipment; these transitions are not simulated at all. We have here processes leading to damages, like shearing, corroding, etc. and also those control procedures whose simulation is not split into a sequence of steps, like turning on the diesel or disengaging the SAC. In all such cases we only simulate the resulting situations. Closely related to transitions are those actions initiated or performed directly by the operator, such as pushing the button which starts the turbine. These actions are simulated in the model.

The above considerations lead us to use four canonical forms of elementary facts, each of which has a set of aspects:

- **Static States.** Apply to EUs only. The aspects of static include: *operational mode* e.g., for a clutch, *engaged* or *disengaged*; *physical condition* e.g., *corroded*, *sheared*; *functional condition* — how the EU behaves when operated (the value of this aspect is usually dependent on the same EU's physical condition; these relationships are part of the general domain knowledge about the equipment).

Static state aspects are directly settable in the simulation model.

- **Dynamic States.** Apply both to EUs and to WSs. The aspects of dynamic states depend on the type of their objects. For system EUs, like turbines or compressors, we use two aspects: *operation* which can take values like *standstill*, *routine running*, *jacking over*, *coasting down*; and *operation modification* which can be used to characterize the operation further by features like *normal*, *unreliable*, *erratic*, etc. For WSs, we have aspects such as *temperature* or *pressure*.

Dynamic states cannot be set directly in the model. They always result from changes initiated by the operator's actions or by accidental changes (most notably damages) of the static states of some EUs. It is however possible to test dynamic states, so we describe these as *testable states*.

- **Transition States.** Apply both to EUs and to WSs denoting changes of their static and dynamic states. It is therefore not surprising that their aspects depend on the type of their objects in the same way as described above for static and dynamic states. The aspect values are modifications of the corresponding aspect values of static and dynamic states. For example, we might have *increasing* as value of aspect *pressure* in a transition state of a WS whose resulting dynamic state would have low as the value for the aspect *pressure*.
- **Actions.** Performed by an abstract operator (whether in reality this is one person or a network of cooperating technicians is of no interest to us) to achieve specific goals defined by the action's aspect. Possible aspects are: *routine operation*, *test*, *repair* and *emergency*. The objects of these actions are EUs or WSs.

In establishing causal relationships we are interested only in those links which involve abnormal situations. Therefore it is necessary to distinguish between normal and abnormal facts. Unfortunately, some facts can be both, depending on the context in which they occurred. In the example of *Pump will not turn when engine jacks over* we cannot decide on the basis of the sole fact of the pump not turning whether the report speaks of a normal or of an abnormal situation. Some facts directly point to abnormal situations, for example alarms, damages or abnormal parameter readings. Others, like operator control operations, are always normal. We have therefore three functionality indicators of facts: *normal*, *abnormal*, and *unknown*.

Because of the importance of the time factor in the reports, facts are also characterized by the intervals during which they are true. In most cases such intervals are not given explicitly in the reports and have to be inferred. In the taxonomy used for this purpose we followed the approach presented in [8] and [17].

Taken together, our representation includes the following information on EFs:

- **predication:** the EF type, its object and aspect as well as other features depending on the particular type, such as the category for Actions and Transition States.
- **time interval:** Interval during which the Predication is true, and its temporal relation to other intervals.
- **functionality:** Normal, abnormal, or unknown.
- **fact evidence:** Default value is the report itself. In some cases, however, the evidence is weakened by the operator's qualifications like *suspect* or *believe*. Some facts from the Discourse Analysis Result are hypothesized by the inference module.
- **causal links** to other facts from the discourse.

7.3 Creating Elementary Facts and Discourse Structures

The input to the Discourse Analysis Module is an ordered set of propositions extracted from the report. There are two kinds of propositions: *events* and *relations*. A sentence can be semantically interpreted as one or more propositions. Each event consists of a predicate and arguments which are either DEs determined by the NP Analyzer or other propositions. Furthermore, attached to each event are the syntactic features of the verb (its tense and aspect) underlying the event proposition. The process of converting events into EFs is governed by the event proposition's predicate. We distinguish two categories of predicates:

1. **First Order Predicates.** A proposition with such a predicate takes nonpropositional arguments only. It gets translated into a temporal structure consisting of one or more EFs, linked by *time arcs* which define temporal precedences. The interpretation depends on the proposition's actual arguments and on the tense and aspect information of the underlying verb. For examples of these data structures as well as the interpretation process we refer the reader to appendix A where an analysis of a report is described in detail.
2. **Higher Order Predicates.** A proposition with such a predicate takes at least one propositional argument. The interpretation of these predicates can be one of the following:
 - (a) **Modification of the elementary fact** which was created as the interpretation of the propositional argument. Most often it is the EF Evidence feature which gets modified in this way. This follows from our design decision not to build any operator's model. Hence, there is no need to make the operator's beliefs into EFs. We treat the belief propositions merely as modifiers of those EFs (namely their Evidence feature) which are objects of such beliefs. Consider for example the sentence: *Suspect faulty high speed rotating assembly*. The *suspect* proposition is not recorded as a separate EF (which reflects our belief that the information about the interval during which the suspicion holds is of no crucial importance to the understanding of the report) but rather as the Evidence type (namely Operator's Suspicion) of the EF describing the damage of the high speed rotating assembly. Perceptual acts dealing with what the operator saw, heard, felt, or smelled, inter-personal communications describing what was reported to the operator by other members of the crew, and monitoring actions reporting what the operator measured or monitored fall into the same category. However, in these cases the Evidence feature of the EF is just the report: from the viewpoint of understanding the

message there is no difference between stating *loud noises coming from diesel* and *heard loud noises coming from diesel*. To report the first, the operator must naturally have heard the noises in the first place.

- (b) Adding new relations between other EFs which were created as the interpretation of the propositional arguments. Consider the sentence *Borescope investigation revealed a broken tooth on the hub ring gear*. Elementary facts are created both for the *investigation* and for the *break* proposition. However, the meaning of the reveal proposition can be restricted to its function as an indicator of how these two elementary facts are related: again, viewed from the perspective of understanding it seems sufficient to interpret reveal by establishing an Evidence link between the two facts. Another example, *Hub failed, causing spline assy to fail causing damage to the SAC* illustrates a proposition (*cause*) which conveys explicit information on the causal relationship between other EFs.
- (c) Changing the structure of a EF group which was created as the interpretation of the propositional argument. Consider the role of *unable* in the sentence *Unable to start the nr 1B turbine*. The EF group created for the *start* proposition contains a Transition State EF for the turbine whose *result* is *completed* and *functionality*: *normal*. However, *unable* changes this interpretation into one where the Transition State of the turbine must have *result*: *arrested before completion* and *functionality*: *abnormal*.

Discussing the translation of propositions into EF groups we tacitly assumed that the events described by them were singular, i.e. they occur only once. This is not always so. Sometimes it is important to convey the repetitive character of the events. Consider a slight modification of the sentence discussed above: *Was unable to CONSISTENTLY start the nr 1B turbine*. The adverb indicates that there were several attempts at starting the turbine and some of them were successful whereas other failed. Because time is an important factor in representing EFs we have a problem: should we include into the discourse representation several instances of the EF group corresponding to the *start* event? It would be difficult to make inferences with such sets. Furthermore, we don't know how many times the event was repeated. The solution we came up with for this problem is a special structure which we call *Habitual*. A habitual consists of one situational EF, a group of behavioral EFs, a time interval called the *time basis*, and a repetition character whose values are keywords like *erratic* or *always same*. The semantics of the habitual is as follows: whenever the time interval of the situational EF falls within the confines of the time basis, the EFs from the behavioral group follow according to the repetition character. Habituals are indicated either directly by words like *consistently*, *repeatedly*, *whenever* or by a combination of words and tenses, e.g. *when* coupled with future or present tenses is equivalent to *whenever*.

After all the input events have been processed, the relations between the EFs are analysed. They have the form (Temporal-Conjunction event₁ event₂), where Temporal-Conjunction may be *before*, *during*, *while*, etc. As a result of their interpretation, some time arcs relating endpoints of the intervals associated with event₁ and event₂ are added to the TIME GRAPH. We defined several relationships between intervals, e.g. (*starts-during* int₁ int₂), (*intersection* int₁ int₂) in terms of how their endpoints are related. Doing this we followed the taxonomy given by [1]. The first step in the interpretation is to decide which EFs are representative for the temporarily conjoined events. Next, a relationship is chosen from a table mapping the products of temporal conjunctions and EF types into interval relationships. Finally, according to the chosen relationship, new time arcs are added.

7.4 Causal Relationships

We have noted above the importance of linking events into causal chains. For some reports it is necessary to build more than just one such chain. Usually they overlap partially. For example, the following CASREP

While diesel was operating with SAC disengaged, the SAC LO alarm sounded. Believe the coupling from diesel to SAC lube oil pump to be sheared. Pump will not turn when engine

jacks over.

has two causal chains: one involving the alarm, the other containing the engine jacking over. The shearing of the coupling is a common starting point for both these chains.

The most important reason for making causal and temporal relationships explicit during understanding is the observation that some of the information we considered important for understanding of CASREPs in section 7.1 can be extracted from causal chains. For example, they help to distinguish between primary and derivative problems as well as between intermediate and ultimate consequences of damages. Sometimes, especially in analyzing failure trends to discover possible design flaws, it may be very important—in cases when two or more damages were reported—to know whether one of them could have caused the other ones.

Building causal and temporal dependencies between reported facts reflects a significant aspect of understanding. We can view a CASREP as a story and following [21] claim that an event in a CASREP is understood only when a plausible explanation for that event with respect to other events in the CASREP is found. The approach of causal and temporal chain building to understanding has been mainly applied to stories about personal encounters, for example [22]. The understanding process was based on a collection of rules capturing knowledge about beliefs and actions of people. For two given facts from a story, a causal link between them was established, if it was possible to construct a path which had the two facts as endpoints and possibly some hypothesized facts as intermediaries. The crucial feature of this path was that any two adjacent facts from it were related by causality expressed by some rule from the knowledge base. The solution we chose for analyzing CASREPs is an extension of this approach. In contrast to the knowledge about people's behavior which is characterized by a relatively large margin of uncertainty, the knowledge on which we base our understanding is in substantial part very precise: many of the events encountered in CASREPs can be simulated in a deterministic way using the PROTEUS equipment model. In the section describing the model we demonstrated that it is possible to propagate consequences of an event throughout the model until a stable state is reached. It means that given two facts as candidates to be causally linked, we can simulate one of them and check whether the other can be confirmed in the resulting stable state of the model. One of the problems with finding causal links is to guess which pairs of EFs should be tried. Considering all pairs would lead to a large number of queries. As observed in section 7.2 we are interested in causal links between abnormal facts only. Furthermore, in presenting the taxonomy of EFs we discussed their settability and testability. Both these features of EFs become useful now.

At the end of the stage during which EFs are created, habituals recognized, and explicit temporal relations processed (as described in section 7.2), the Discourse Analysis Result has the form of a directed acyclic graph, usually unconnected, consisting of several connected subgraphs. We say that the EFs which are part of such subgraphs describe *situations*. The next step in the analysis is to decide which situations are abnormal. When at least one of the constituent EFs is abnormal, we classify the situation as abnormal. A situation is considered normal when all its constituent EFs are normal. Other cases are those when there are no abnormal EFs and at least one has the *functionality: unknown*. In such cases we resort to simulation. Consider again the example introduced in section 7.2, *Pump will not turn when engine jacks over*. To classify this situation we first create normal operational conditions for the SAC and then run a simulation test which consists of jacking over the engine and checking whether the pump is turning. If we find the pump turning, we can conclude that the situation described in the sentence is abnormal.

Once the abnormal situations have been found, we take all possible ordered pairs from among them and for each one try to determine (using simulation) whether its elements can be causally related. Let's call a pair's first element an *antecedent* and the second one a *consequent*. If any of the antecedent EFs is a testable fact or if there is no testable fact among the consequent EFs, we discard such a pair from further considerations. Otherwise we run two simulation tests. The first one consists of the following steps:

1. all the settable EFs from the antecedent as well as the settable EFs preceding the testable one from the consequent are set in the model,
2. if there are some remaining parameters whose setting is necessary for the SAC to operate in the mode indicated by the antecedent or consequent situation, they are set to default values,
3. the scenario resulting from the above steps is simulated,
4. the condition described by the abnormal testable EF (there may be more than one abnormal EF in the consequent, but only one among them should be testable) is checked.

The second test is similar, the only difference being that instead of setting the abnormal EF from the antecedent (again, there may be more than one abnormal EF in the antecedent, but only one among them should be settable), we set its normal counterpart. Now, if we get `true` as a result in the fourth step in the first test and `false` in the same step in the second test, we conclude that there is a causal link between the abnormal EF from the antecedent and the abnormal fact from the consequent. Any other pair of results doesn't justify such a conclusion.

A possible heuristic for constructing useful simulation queries (not implemented yet) consists of grouping abnormal facts from class Dynamic State into sets corresponding to the WS media on which the damaged or malfunctioning EUs operate. Each such set contains facts which are good first candidates for arguments of causal relationships.

The ultimate goal of this part of the understanding procedure is to include all abnormal EFs into some causal chain. When it is not possible to achieve this, attempts are made to hypothesize facts and treat them as additional elementary facts for the purpose of building more elaborate causal chains which would possibly include the "loose" facts. The hypotheses are based on general domain knowledge about equipment, and especially its failures. This knowledge is organized as a set of IF—THEN rules. It is important to notice here that the inferred fact is only *possibly* true. A hypothesis is supported if it can be successfully used to link two original facts into a causal chain.

Once the causal links have been identified, new time arcs can be added to the TIME GRAPH. They follow from a straightforward observation about causal links: if a fact EF1 has been discovered to be a cause for another fact EF2, then the starting point of the time interval during which EF1 is true must precede the starting point of a similar interval for EF2.

7.5 Model Query Processor

In section 6.4 we showed how queries to the model were helpful for noun phrase analysis. Here we concentrate on model queries related to the discourse analysis. We differentiate between dynamic *simulation queries* and static *structural queries*.

The simulation queries have one of the three forms:

- **NQ-Simulation-Set** (*ef-proposition*)
- **NQ-Simulation-Test** (*ef-proposition*)
- **NQ-Simulation-Defaults** (*eu, mode*)

where *ef-proposition* is an EF proposition, *eu* is an EU and *mode* is one of its possible operational modes. The first two queries are self-explanatory, the third is the one used in the third step of a causal link simulation test described above. **NQ-Simulation-Test** is a predicate.

Structural queries are used in IF—THEN rules for checking the context in which they may be applied. Most of the context constraints can be expressed by the predicate verification queries which are used for the noun phrase analysis (cf. section 6.4.3) but some additional query forms were required. One structural query which was introduced primarily for discourse analysis examines paths between model nodes. It has the form:

```
(MQ-Path from-eu to-eu
  &opt :from-medium :to-medium :always-medium)
```

from-eu and to-eu are two different EUs in the model. from-medium, to-medium and always-medium are optional arguments which in our domain take one of the values: oil, air, or rotation. MQ-Path returns false if there is no path in the model network between the nodes at which from-eu and to-eu reside. Otherwise, it returns the first path found (there may be more!). The optional arguments put constraints on the links. If from-medium is specified, the WS associated with the first link in the path must have the given medium as part of its description. If to-medium is specified, the WS associated with the last link in the path must have the given medium as part of its description. If always-medium is specified, the WSs associated with all the links in the path must have the given medium as part of their descriptions. An example of an IF—THEN rule which uses MQ-Path queries in its context part is:

```
if (Static-State :equip-unit *1 :aspect phys-cond :value seized)
  in context of
    ((MQ-Feature :equip-unit :equip-unit *2
      :function generate :medium rotation)
     (MQ-Path *2 *3 :always-medium rotation)
     (MQ-Path *3 *1 :always-medium rotation))
  then (Static-State :equip-unit *3 :aspect phys-cond :value sheared)
```

The interpretation of this rule is: *If an element is seized, then it is possible that an element lying on the path of rotary elements between a generator of the rotation and the seized element shears.*

7.6 Report Summary

The last stage of the discourse analysis creates a report summary which reflects the purposes of analyzing CASREPs - they can be different depending on where and to whom the reports are dispatched (cf. section 7.2). The information which gets into the summary is extracted from the structure containing everything which PROTEUS was able to understand (the collection of EFs and their relationships). For each specific application there exists a different summary prototype designed in such a way that the most crucial information (from the perspective of a given application) will be extracted. For example, a summary might contain tables of damage, functionally impaired equipment, and corrective actions. Each damage entry would include the industrial name of the damaged element, the place of the element in the modeled equipment, the nature of the damage, the degree of certainty that the damage occurred, and the information on whether the damaged element had been fixed, replaced or left in place. The impaired equipment and corrective action entries would include similar details. This part of the system hasn't been implemented yet.

8 Equipment Model Editor

The PROTEUS system presented above, at its present stage of development, can only process reports about malfunctions of the starting air system. Considering that this system constitutes only a tiny portion of the entire range of shipboard equipment covered by CASREPs, we cannot claim at present to have a very practical text processing system.

If technical text understanding systems like PROTEUS, based on equipment simulation, are to be of any practical value, techniques must be provided to enable users other than the designers of such systems to tailor or expand them to new domains. The most difficult part of this task is building simulation models for the equipment in the new domains. Being aware of all this from the very beginning of our research, we have made many design decisions to facilitate the future development of a module for the computer-assisted design of new equipment models.

Even now creation of a model doesn't require specification of every slot in every single instance in the model. In the first place, many of the instance slot fillers are derived from information in the prototypes. In addition, we have developed a quite elaborate initialization procedure whose task it is to build a fully-fledged model from specifications expressed in a more concise and natural way than meticulous instance definitions. In particular, we require at present that only instances describing EUs be given explicitly. Some slots of these instances contain information for the initialization procedure. The most notable examples here are slots describing how the EU modeled by an instance is linked to other EUs on the same and on the immediately higher levels. This information is used by the initialization procedure to create instances of appropriate links. The information about links to be found in these slots is accompanied by information specifying the constant features of the WSs which are supposed to be associated with the links. The instances of these WSs are built automatically by the initialization procedure as well.

A very important resource at our disposal which can dramatically improve the model building process is our graphical display. The initialization procedure we have developed so far can provide the basis for a powerful equipment model graphic editor. We have been aware of this possibility from the very beginning and have fashioned the graphical display with this potential application in mind. For example, all of the model network icons for nodes and links have been written into a regular grid which can be made visible on demand.

A graphical editor would allow us to replace the current approach of coding the individual EU instance specifications with an interactive session. At the beginning of the editor session an empty grid would be displayed along with a menu of icons for the prototype equipment like valves, pumps, filters, gears, etc. The user could then drag any of these icons into a chosen place on the grid. This done, the editor would involve the user in a menu-driven dialogue to capture the specific information needed to create an instance of the prototype corresponding to the chosen icon. In a similar fashion the user could specify all the links (i.e. dragging their icons into appropriate places and providing specific information by means of a dialog). The information necessary to complete the descriptions of the WS associated with links need be given only at the nodes which are sources of this kind of WS. The setting of nodes and links would be repeated until the user had completed an entire network. The user would then choose one of the displayed nodes for recursive refinement. This stepwise top-down process could continue until the entire model had been specified.

This scenario assumes that the whole model is built from instances of predefined prototypes. It is more realistic to assume that it will be possible to piece together only a portion (although probably quite substantial) of a new equipment from predefined prototypes. Therefore, it will be necessary to facilitate defining new prototypes as well. This will probably require a more thorough knowledge of the system on the part of the user.

9 Implementation

The PROTEUS system has been substantially implemented and debugged and has been publically demonstrated operating on a small set of actual CASREPs. Of the components described in detail above: the equipment model has been fully implemented as described. The noun phrase analyzer has been fully implemented except for the mechanism which would handle *superstructures* (cf. section 6.4.2) and the contributions of clause semantics and discourse analysis to reference resolution. For discourse analysis we have implemented everything except the generation of report summaries (cf. section 7.6).

10 Conclusion

CASREPs are written on board Navy ships and sent to various Navy commands for collection and analysis. PROTEUS has been designed for use by the recipients of these messages, in order to automate message processing. In these concluding remarks we want to investigate an attractive extension to this

application. As we have seen in this paper, the approach we have taken is based on a rather detailed simulation of the equipment which is the subject of the CASREPs to be analyzed. The simulation model is the *world* in which we interpreted CASREP texts. Looking at reported facts in the context of this world allowed us to solve some difficult understanding problems, most notably identifying concepts referred to in the reports and finding coherence relationships between their individual sentences. This was possible because the simulation model proved to be a rich source of information about the domain as well as a good inference mechanism. It was also a useful tool for checking hypotheses. In effect, the model together with the query processor operating on it could be justifiably referred to as a small expert system.

We believe that expanding PROTEUS to a powerful interactive diagnostic assistant would be a logical consequence of the approach we took to language understanding. Then it would be possible to deploy PROTEUS systems not only where the CASREPs are collected but directly on Navy ships. To see why this should be desirable let us make some general comments on CASREPs. Reading them we got the feeling that the degree of effort (later reported) and expertise involved in attempts to locate reasons for problems differed greatly from case to case. At one extreme there are cases when nothing was done to even identify the nature of the problem. On the other hand there are quite long reports (too long to be quoted here) which describe successful diagnostic sessions leading to the full identification of the causes of failure. In between are CASREPs which describe symptoms of problems and then:

- only speculate on the possible damage which could explain these symptoms (*It is likely the LO pump has sheared.*), or
- give indirect evidence without indicating any damage (*Start air pressure dropped below 30 psig during monitoring of 1A turbine. Oil is discolored and contaminated with metal.*), or
- postpone the diagnosis until more tests have been conducted (*Retained oil sample and filter element for future analysis.*), or
- give no reasons for the occurrence of the symptoms at all (*During routine start of main propulsion gas turbine, SAC air pressure decreased rapidly to 5.74 psi resulting in an aborted engine start. Exact cause of failure unknown.*).

These wide variations suggest that, if only the ship technicians were better experts, the level of diagnosis would have been deeper and the CASREPs correspondingly more informative.

Because it is unrealistic to expect that all technicians possess the same high level of expertise, it could be helpful to have an interactive diagnostic assistant on the ship which could lead a relatively inexperienced technician through a diagnostic session. This guidance would take the form of suggestions for tests to be performed and requests to the technician for more diagnostic information. The success of such a system would depend greatly on the ease with which the initial symptoms could be described and the quality of the subsequent interactions. Because of the wide variety of possible symptoms (as evidenced by the CASREPs), a combination of a rich natural language capability and a graphic interface seems called for. We believe therefore that PROTEUS, with its integrated language understanding, graphic interface, and detailed equipment model, is particularly well suited to be the base for such a system.

11 Acknowledgments

This paper is based upon work supported by the Defense Advanced Research Projects Agency under Contract N00014-85-K-0163 from the Office of Naval Research, and by the National Science Foundation under grant DCR-85-01843. We wish to acknowledge the contributions of: Leo Joskowicz, who developed some of the causal analysis procedures and assisted in the final preparation of this manuscript; John Sterling, who developed the current implementation of the noun phrase analyzer and wrote parts of appendix A; Ngo Thanh Nhan, who developed the grammar and implemented the clause analyzer; and Michael Moore, who integrated the components on the *Symbolics*.

References

- [1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 832-843, 1983.
- [2] D. Bobrow, R.M. Kaplan, M. Kay, D.A. Norman, H. Thompson, and T. Winograd. Gus, a frame-driven dialog system. *Artificial Intelligence*, 155-173, 1977.
- [3] D. Bobrow and T. Winograd. An overview of krl, a knowledge representation language. *Cognitive Science*, 1-46, 1977.
- [4] D.G. Bobrow, editor. *Qualitative Reasoning about Physical Systems*. The MIT Press, 1985.
- [5] T. Bylander and B. Chandrasekaran. Understanding behavior using consolidation. In *Ninth Int. Joint Conf. on Artificial Intelligence*, pages 450-454, 1985.
- [6] E. Charniak. On the use of framed knowledge in language comprehension. *Artificial Intelligence*, 225-265, 1978.
- [7] J. de Kleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 7-83, 1984.
- [8] D.R. Dowty. The effects of aspectual class on the temporal structure of discourse: semantics or pragmatics? *Linguistics and Philosophy*, 37-61, 1986.
- [9] T. Finin. Nominal compounds in a limited context. In R. Grishman and R. Kittredge, editors, *Analyzing Language in Restricted Domains*, pages 163-173, Lawrence Erlbaum, 1986.
- [10] K.D. Forbus. Qualitative process theory. *Artificial Intelligence*, 85-168, 1984.
- [11] R. Grishman, T. Ksiezyk, and N.T. Nhan. *Model-based analysis of messages about equipment*. Technical Report 236, Computer Science Department, New York University, 1986.
- [12] T. Ksiezyk, R. Grishman, and J. Sterling. An equipment model and its role in the interpretation of noun phrases. In *Tenth Int. Joint Conf. on Artificial Intelligence*, pages 692-695, 1987.
- [13] B. Kuipers. Commonsense reasoning about causality: deriving behavior from structure. *Artificial Intelligence*, 169-203, 1984.
- [14] W. Lehnert. Plot units: a narrative summarization strategy. In W.G. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing*, pages 375-412, Lawrence Erlbaum, 1982.
- [15] E. Marsh, H. Hamburger, and R. Grishman. A production rule system for message summarization. In *Proc. of the Nat. Conf. on Artificial Intelligence*, pages 243-246, 1984.
- [16] M. Palmer, D. Dahl, R. Schiffman, L. Hirschman, M. Linebarger, and J. Dowding. Recovering implicit information. In *24th Annual Meeting of the ACL*, pages 10-19, 1986.
- [17] R. Passonneau. A computational model of the semantics of tense and aspect, forthcoming. *Computational Linguistics*, 1988.
- [18] C. Rieger. An organization of knowledge for problem solving and language comprehension. *Artificial Intelligence*, 89-127, 1976.
- [19] D.E. Rumelhart. Understanding and summarizing brief stories. In *Basic processing in Reading. Perception, and Comprehension*. Hillsdale, 1977.

- [20] N. Sager. Natural language information formatting: the automatic conversion of texts to a structured data base. In M.C. Yovits, editor, *Advances in Computers*, Academic Press, 1978.
- [21] R. Schank and R. Abelson. *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum, 1977.
- [22] R. Wilensky. *Planning and Understanding. A Computational Approach to Human Reasoning*. Addison-Wesley, 1983.

A Noun Phrase Analysis Example

As an illustration of some considerations presented in section 6 we present the analysis of a particular noun phrase (NP):

starting air temperature regulating valve

The *Syntactic Analyzer* produces the following structure

```
(NP valve
  (AN-STG ( (progressive start)
            (air singular)
            (temperature singular)
            (progressive regulate))))
```

Note that the parser has grouped the pre-nominal adjectives and nouns (AN-STG) into a list but not assigned any further structure. This structure is passed through *Clause Semantics* and becomes input to the *NP Analyzer*. The verbs *start* and *regulate* map into the predicates *Start* and *Regulate* respectively. The semantic classes of the non-predicate items are: *air* — substance, *temperature* — feature, *valve* — equipment. Using the predicates and the semantic classes the NP Analyzer tries to map the words into a nested structure of patterns as discussed in section 6.3. Each such pattern is assigned a semantic class so that it may become a member of a higher-level pattern. Thus, *starting air* is mapped into a pattern whose class is substance. All possible analyses are constructed bottom-up using these patterns. One possible interpretation for our NP is:

```
(Regulate :head :instrument
          :instrument valve
          :object (Measure :head :feature
                          :feature temperature
                          :object (Start :head :instrument
                                       :instrument air)))
```

This interpretation corresponds to the parse:

```
((((starting air) temperature) regulating valve)
```

Along the way, a pattern corresponding to another parse

```
((air temperature) regulating valve)
```

will also be formed. However, it will not play a part in the final analysis, because the class of this phrase, equipment, does not appear at the *:instrument* position in any pattern headed by *start*.

The analysis is not unique. The one corresponding to:

```
((starting air) (temperature regulating valve))
```

will also be formed. The interpretation here is:

A valve, which regulates temperature (of something unspecified) and also operates (in an unspecified manner) on air used for starting.

There is no mechanism provided at present for preferred readings. Hence, no preference is given between these two interpretations. Both will be used to construct queries to the domain model. And both will, in this case, result in the same valves being identified. However, in other examples, an analysis generated by the *NP Analyzer* can be disallowed by the pragmatics of the domain model.

Once all the interpretations are formed, the *NP Analyzer* recursively invokes the *Model Query Processor* in order to determine the DE(s) which can be the referent of the NP. Space considerations allow us to present here only the queries used for the first interpretation:

- for valve, (MQ-DE-Request valve) returns the set of all the valves in the model:
(valve₁, valve₂, ..., valve₇);
- for temperature, (MQ-DE-Request temperature) returns a newly created DE (let its generated name be wprop₃₉), of class *medium property* (cf. section 6.4.1);
- for air, (MQ-DE-Request air) returns a newly created DE, (let its generated name be wdesc₂₇), of class *medium* with *medium part* set to air (cf. section 6.4.1);
- for the phrase *starting air*,
(MQ-Pred-Verif Start :head :instrument :instrument wdesc₂₇),
returns wdesc₂₈ which is a modified version of wdesc₂₇ (the function part of wdesc₂₈ is filled with Start);
- for the phrase *starting air temperature*,
(MQ-Pred-Verif Measure :head feature :feature wprop₃₉ :object wdesc₂₈),
returns wprop₄₀ which is a modified version of wprop₃₉ (the property name part of wprop₄₀ is filled with temperature);
- finally, for the entire NP, the query

```
(MQ-Pred-Verif Regulate :head instrument
                        :instrument (valve1, valve2, ..., valve7)
                        :object wprop40)
```

is generated. For each of the valve_{*i*} the *Model Query Processor* tests in the model whether one of the valve's functions is to regulate a WS whose description is given in wprop₄₀, i.e. whether

1. its medium is air and
2. there is any EU (eu_{*x*}) in the model such that there exists a path from valve_{*i*} to eu_{*x*} with
:always-medium = air (cf. MQ-Path, section 7.5) and eu_{*x*} is started by the medium associated
with the last link of the found path and
3. the regulated parameter is temperature.

In the model only one of the valves, namely valve₄, survives this test. Hence, valve₄ is returned as the only DE for the NP.

B Discourse Analysis Example

We show the analysis of the following message:

While diesel was operating with SAC (Starting Air Compressor) disengaged, the SAC LO (Lubricating Oil) alarm sounded. Believe the coupling from diesel to SAC lube oil pump to be sheared. Pump will not turn when engine jacks over.

The *Clause Semantics Module* (cf. Fig. 1) produces the following representation for the message which is kept in a global variable **context**:

```
( (EVENT e-3 (Is-Disengaged sac1))
  (EVENT e-4 (Operate diesel1))
  (TNS-ASP e-4 (PAST PROGRESSIVE))
  (EVENT e-5 (Sound alarm1))
  (TNS-ASP e-5 (PAST))
  (EVENT e-6 (Is-Sheared coupling1))
  (EVENT e-7 (Believe ANYONE e-6))
  (TNS-ASP e-7 (PRESENT))
  (EVENT e-8 (Jack-Over diesel1))
  (TNS-ASP e-8 (PRESENT))
  (EVENT e-9 (Not (Turn pump1)))
  (TNS-ASP e-9 (FUTURE))
  (RELATION r-4 (WITH e-4 e-3))
  (RELATION r-5 (WHILE e-4 e-5))
  (RELATION r-6 (WHEN e-9 e-8)))
```

where *sac₁*, *diesel₁*, *alarm₁*, *coupling₁* and *pump₁* are names of DEs determined by the *NP Analyzer* in a similar way as described in appendix A.

The analysis of this input can be split into stages:

(1) For each event from **context** with a first-order predicate (cf. section 7.3) its translation into EFs (see also Fig. 4) is created:

```
For e-3: EF1
For e-4: EF2
For e-5: EF3, EF4, EF5
For e-6: EF6
For e-7: EF7
For e-8: EF8
```

Note that *alarm sounded* is treated as a change of state, from *alarm quiet* (EF3), through a transition (EF4), to *alarm sounding* (EF5). The description of an EF consists of the parts shown in section 7.2. Below we show the EF predications (in a slightly simplified form, omitting some of the arguments irrelevant for this presentation):

```
EF1:   (Static-State      :equip-unit sac1
      :aspect OPERATIONAL MODE
      :value DISENGAGED)

EF2:   (Dynamic-State     :equip-unit diesel1
      :aspect OPERATION
      :value NORMAL)
```

```

EF3:   (Dynamic-State :equip-unit alarm1
        :aspect OPERATION
        :value NORMAL)
EF4:   (Transition-State :equip-unit alarm1
        :aspect OPERATION
        :value ACTIVATING)
EF5:   (Dynamic-State :equip-unit alarm1
        :aspect OPERATION
        :value SIGNALING)
EF6:   (Static-State :equip-unit coupling1
        :aspect PHYSICAL COND
        :value SHEARED)
EF7:   (Dynamic-State :equip-unit pump1
        :aspect OPERATION
        :value STANDSTILL)
EF8:   (Dynamic-State :equip-unit diesel1
        :aspect OPERATION
        :value JACKING OVER)

```

The only case deserving a comment is EF7: when the predicate Turn is applied to simple node EU (such as a shaft), it is translated into a dynamic state involving the SPEED of a WS. When it is applied to a system node EU (such as a pump), only a portion of which actually turns, it is translated into a state involving the OPERATION of the EU.

(2) Higher-order predicates from *context* are processed. Only e-7 qualifies as such. No new EF is created; rather, the evidence part of EF6 is set to OPERATOR'S BELIEF.

(3) The RELATION entries from *context* add new elements to the DA Result:

- Interpretation of r-4 introduces a new EF, EF9. Its predication is:

```
EF9: (And :arg1 EF1 :arg2 EF2).
```

It is related to EF1 and EF2 by the time arcs 1→15, 3→15, 16→4 and 16→2; i.e., its time interval is within those for EF1 and EF2.

- Interpretation of r-5 results in adding the time arcs 15→6 and 7→16; i.e., the transition occurs within the interval of EF9.

- The case of r-6 is the most interesting. First, arcs 11→13 and 14→12 are created. Then, on the basis of the tenses associated with e-8 and e-9, and the temporal conjunction of r-6 (when), the situation described by r-6 is recognized as a repetitive one. Hence a habitual (cf. section 7.3), HB1 is created. Its features are:

```
situ-of: EF7; behav-of: (EF8); repet-char: ALWAYS SAME
```

(4) At this point the TIME GRAPH consists of 3 connected subgraphs which constitute the *situations* (cf. section 7.4):

```
SITU1 — nodes: 1, 2, 3, 4, 15, 16, 5, 6, 7, 8
```

```
SITU2 — nodes: 9, 10
```

```
SITU3 — HB1
```

SITU1 is easily recognized as abnormal because of the explicit abnormality of EF4 and EF5. Similarly, because of EF6, SITU2 is abnormal. The case of HB1 had been discussed in section 7.4. The following queries are used here:

```
(NQ-Simulation-Set <proposition of EF8>)
```

```
(NQ-Simulation-Defaults sac1 NORMAL)
```

(MQ-Simulation-Test \langle proposition of EF7 \rangle)
 The result false leads to the conclusion that HB1 is abnormal.

(5) Now PROTEUS is ready to look for causal links using the simulation mechanism. Applying the procedure delineated in section 7.4 two pairs of tests are run:

(1a):
 (MQ-Simulation-Set \langle proposition of EF6 \rangle)
 (MQ-Simulation-Defaults sac₁ NORMAL)
 (MQ-Simulation-Test \langle proposition of EF5 \rangle)
 (1b):
 (MQ-Simulation-Defaults sac₁ NORMAL)
 (MQ-Simulation-Test \langle proposition of EF5 \rangle)
 (2a):
 (MQ-Simulation-Set \langle proposition of EF6 \rangle)
 (MQ-Simulation-Set \langle proposition of EF7 \rangle)
 (MQ-Simulation-Defaults sac₁ NORMAL)
 (MQ-Simulation-Test \langle proposition of EF8 \rangle)
 (2b):
 (MQ-Simulation-Defaults sac₁ NORMAL)
 (MQ-Simulation-Test \langle proposition of EF8 \rangle)

The results: (true, false) for (1) and (true, false) for (2) justify the inclusion in the DA Results of two causal links: EF6 \Rightarrow EF4 and EF6 \Rightarrow HB1.

(6) Using the dependency between causal and temporal relationships, two more time arcs are added to the TIME GRAPH: 9 \rightarrow 6 and 9 \rightarrow 17. The final version of the DA Result's TIME GRAPH is shown in Fig. 4.

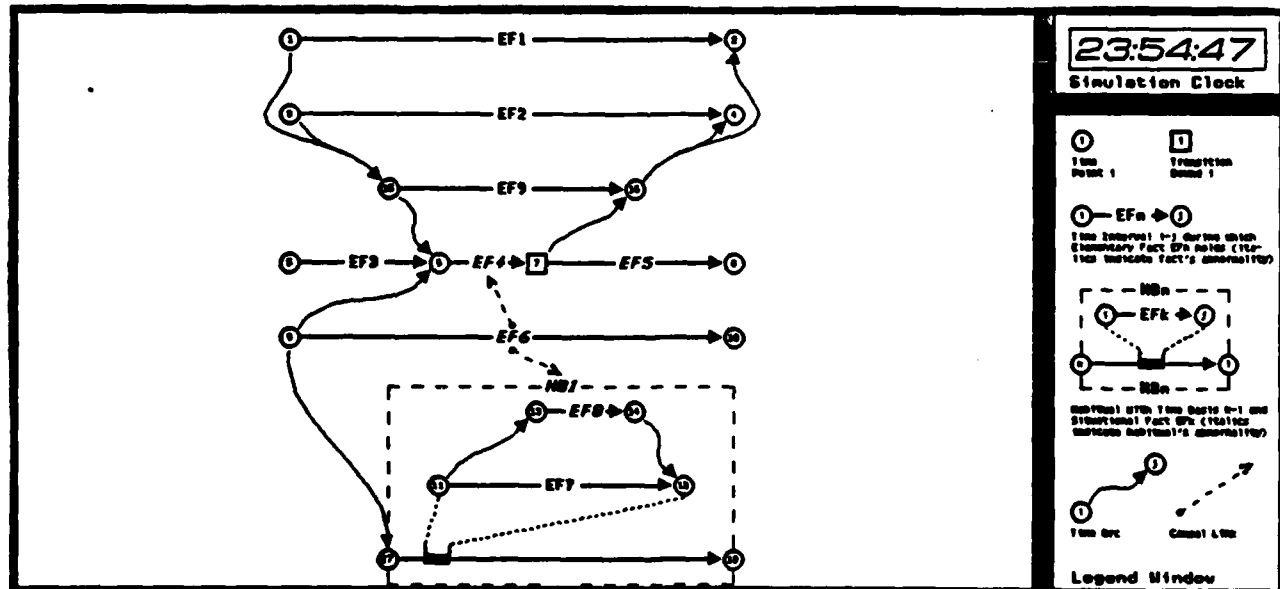


Figure 4: Final Version of the DA Result's TIME GRAPH.